Neural Networks with Several Datasets

Introduction and Objective

Task 1

The first task is to write a program to load the CIFAR10 dataset, preprocess it and train a neural network that predicts the class of the images.

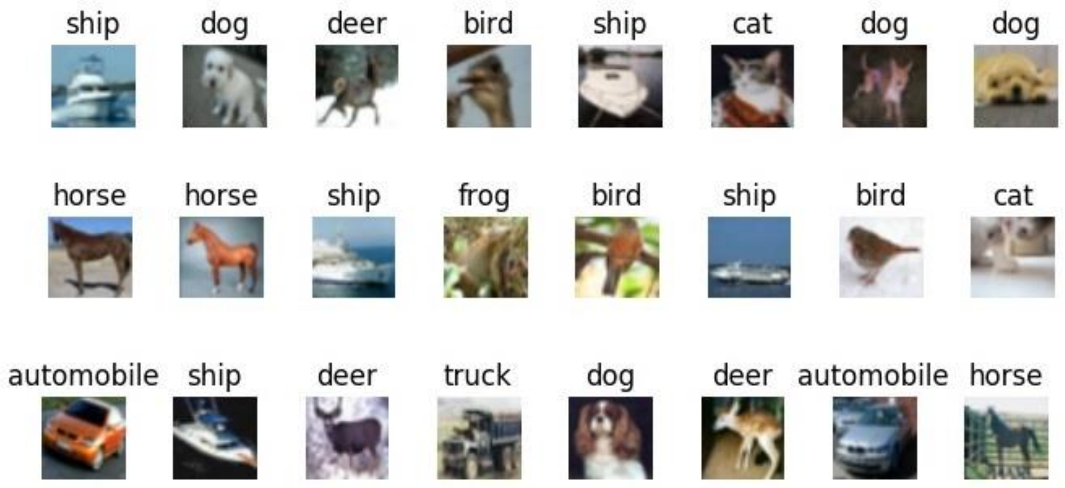


Figure 1 CIFAR 10 image examples

Task 2

The second task is based in the spambase data repository from UCI. This is a big dataset where some characteristics of real emails are given, and we need to classify them being spam or not spam. The difficulty of this problem is the tradeoff between false positives (predicting it is spam and it is not) and false negatives (predicting a non-spam when it is spam).

Task 3

The final task works again with a dataset of the UCI repository, this time the communities and crime dataset will be used. This is a regression problem because the target is the ratio of violent crimes per population.

In this case the difficulty remains in the empty or unknown data. There are many features with unknown values and there are many methods to overcome this problem, filling it with zeros, with the mean of the values of that feature, removing that feature or removing that data example. The best solution depends on the problem, there is not an ideal solution for every problem.

Methods

Task 1

We are going to select three of the ten classes for this problem. Function get_class_i will select the classes that the user defines and from there, we will use those predictive values to train our algorithm.

Then, values of y_train, the class value we want to predict, will be transformed to categorical data. X_train, or the images itself, will be mapped from [0-255] to [0-1] so that the data is normalized.

The architecture of the neural network has been one of the most complicated decisions on this problem. Since in this problem we are only allowed to use fully connected layers, the first option is to use a simple NN:

```
model = Sequential()
model.add(Flatten())
model.add(Dense(512,input_shape=(32,32,3),activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
```

First, we need to flatten the images to be the input of a fully connected dense layer. ReLu activation function will be used in all the layers except in the last one. The last layer needs a softmax activation function because it needs to output the probability of being one of each class.

Then, in a second try, more layers of bigger sizes were added. It has been seen that deeper neural networks work better with this kind of problems even though the cost of computation on the training is higher. However, in this case performance of the model is not increasing that much but the computation cost is higher. So the final decision is to continue with the simple architecture.

An early stopping callback was added in order to stop the training once the validation accuracy stops to increase, therefore, a very high number of epoch will be defined so that the model stops whenever it needs. We use the categorical cross entropy as a loss function since it is adequate for classification problems.

Finally, some hyperparameters were optimized using a grid search: (batch size, epochs, optimizer and its hyperparameters). GridSearchCV from sklearn library was used to search those hyperparameters.

Task 2

In this task we will create a function to load the dataset using the url. After doing it, we are going to transform de csv file to dataframe and then to a numpy array. We could use the dataFrame as the input of the model but for now we will use the numpy array, however, dataFrame will be used in task 3.

The split proportion between train, validation and testing data will be 0.7/0.15/0.15 in this problem. As a part of the data preprocessing, normalization of the features and transformation to categorical of the output class will be performed.

Then, the neural network architecture needs to be defined. We will repeat the strategy of the first task, we will start with a simple network to measure its capacity to classify the correct outputs. We will continue with a more complex architecture to see if the accuracy is improved.

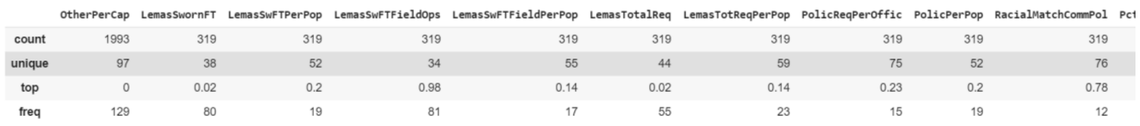
Finally, once the architecture is defined, we will use a gridsearch to find the best values for the hyperparameters.

Task 3

In the last task we need to load the data from the repository. This time we will be using pandas to preprocess the data, so we do not need to transform the dataframe into a numpy array. In order to have the data well organized we will import the headers of the dataset from the same repository.

In this dataset there are five non-predictive features that will be removed for obvious reasons. Then, we will replace cells with a '?' value and insert a np.nan value to clarify there is not any number in that point.

If we plot a table of the number of missing values for each feature:



	OtherPerCap	LemasSwornFT	LemasSwFTPerPop	LemasSwFTFieldOps	LemasSwFTFieldPerPop	LemasTotalReq	LemasTotReqPerPop	PolicReqPerOffic	PolicPerPop	RacialMatchCommPol	Pct
count	1993	319	319	319	319	319	319	319	319	319	
unique	97	38	52	34	55	44	59	75	52	76	
top	0	0.02	0.2	0.98	0.14	0.02	0.14	0.23	0.2	0.78	
freq	129	80	19	81	17	55	23	15	19	12	

Table 1 Missing values

We can see that except OtherPerCap that has one missing value, there are many features with many missing values.

The decision dealing with missing values is to use the mean to replace the one and only missing value in OtherPerCap and to remove all the other features with missing values.

After the elimination, there are 100 features left which is a big number and therefore, we need a more complex network to be able to deal with data of big dimensionality.

Since we need to apply a k fold cross validation method, we will use sklearn library to do so. There will be 4 splits, a stochastic gradient descent optimizer with momentum will be used and the architecture of the net will be:

```python
model = Sequential()
model.add(Dense(512,input_shape=(100,),activation='relu'))
model.add(Dense(512,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
model.add(Dense(1,activation='linear'))
```

First, the input shape is equal to the number of features. In total, there are seven layers with decreasing number of neurons. Since this is a regression problem, the last layer has only one linear output.

As a loss function we cannot use the same loss function as in the classification, therefore we ill use the mean squared error as loss and as the metrics to evaluate the model.

Results

Task 1

The first grid search decides the number of epochs and the batch size. These works trying different combinations of the variables batch_size = [40,50,80,100] and epochs = [25,50,100,150].

The second search consists on finding the best optimizer. These are the results obtained fitting the model with different optimizers and calculating the accuracy, a cross validation with 3 splits was performed:

```
Best: 0.799933 using {'optimizer': 'Adam'}
0.760733 (0.026754) with: {'optimizer': 'SGD'}
0.756933 (0.009648) with: {'optimizer': 'RMSprop'}
0.337067 (0.003364) with: {'optimizer': 'Adagrad'}
0.760467 (0.037439) with: {'optimizer': 'Adadelta'}
0.799933 (0.005290) with: {'optimizer': 'Adam'}
0.789000 (0.005028) with: {'optimizer': 'Adamax'}
0.756000 (0.018874) with: {'optimizer': 'Nadam'}
```

Adam is the optimizer with the best results; therefore, we will continue the search using that. Now that we are optimizing the learning rate, we will fix all the other hyperparameters and focus on trying different values of LR.

```
0.797933 (0.001934) with: {'learn_rate': 0.0001}
0.783933 (0.001112) with: {'learn_rate': 0.001}
0.333333 (0.003974) with: {'learn_rate': 0.005}
0.330267 (0.001517) with: {'learn_rate': 0.01}
```

It can be seen that lower learning rates can reach a higher accuracy, this is because high rates are not able to converge in an optimal point. However, very low rates need many iterations no reach the minimum cost point.

These are the selected hyperparameters for the final model:

1) Batch size: 50
2) Epochs: 50
3) Optimizer: Adam
4) Learning rate: 0.0001

And the results training the final model and evaluating with the test set:

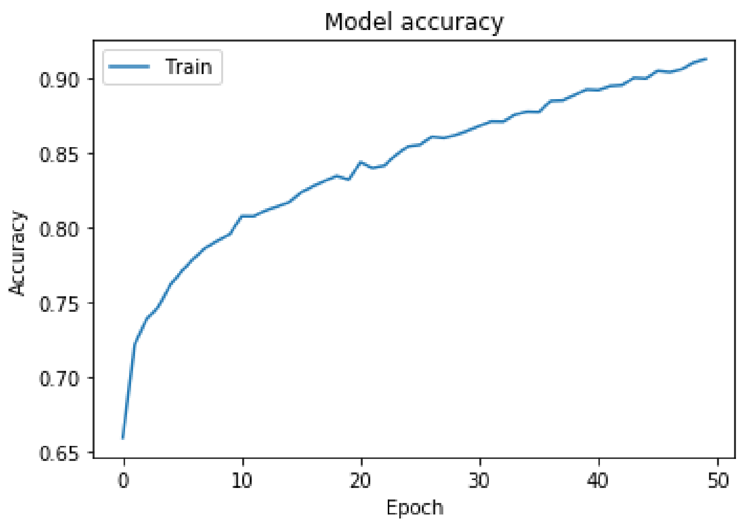


Figure 2 Accuracy of the CIFAR10 model

Finally, test accuracy is 82% which show us that the model could be overfitted because the training accuracy is of 91%.

Task 2

In this dataset there are 56 predictive features, that is, we have far less input variables than in the previous problem. Therefore, the model should be simpler so the number of layers and neurons will be reduced.

I trained the first model after applying normalization and obtained these results:

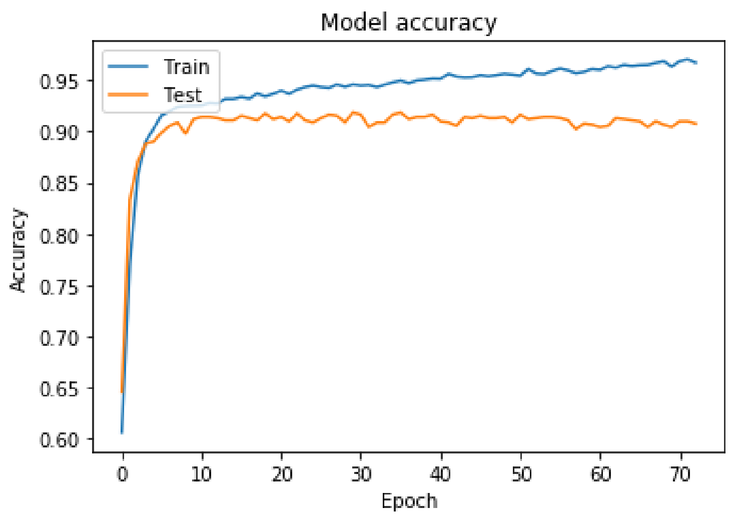


Figure 3 Accuracy of the model

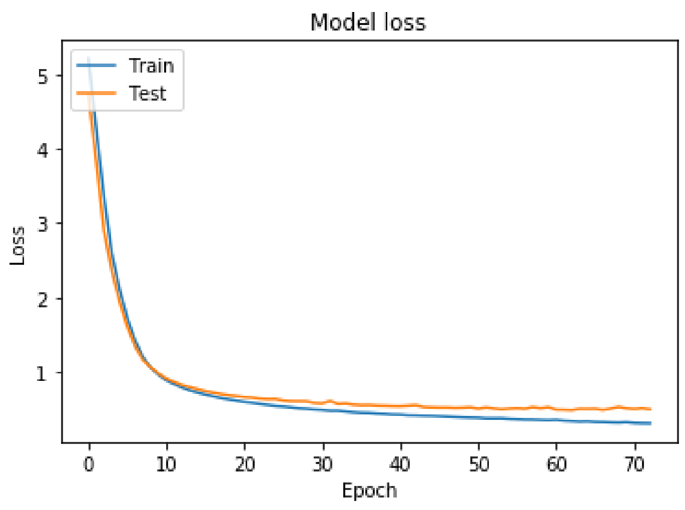


Figure 4 Model loss

This model is overfitting the data since the training accuracy is greater than test accuracy. In order to solve that and obtain a greater validation accuracy, we are going to use l1 regularization. After using those kernel regularizers, the model is less overfitted. The complexity of the network is also another variable that affects the overfitting but our model is quite simple so I do not think it is necessary to reduce it. In the next case we are going to increase the batch size and see how the model performs:

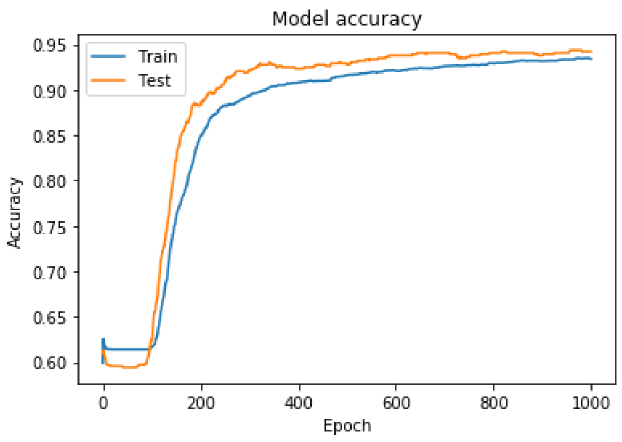


Figure 5 Accuracy of the second model

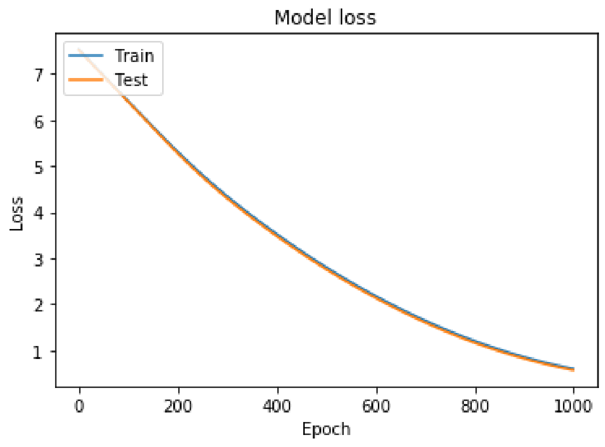


Figure 6 Loss of the second model

Validation accuracy has been increased from 91% to 94% and the model is not overfitting the training data more.

Finally, the last step is to evaluate the model in testing data and doing so we get and accuracy of 94.2%.

Task 3

In this last task, there are 100 predictive variables so the network we will build will be quite larger than the one of the previous tasks. We will be using Adam optimizer and a mse loss function.

The program also plots the correlation matrix of all the variables, this can help us understand which features are helpful to predict the crimes per population. Also, it can be seen that some features that are highly correlated are actually redundant, because they are giving the same information twice.

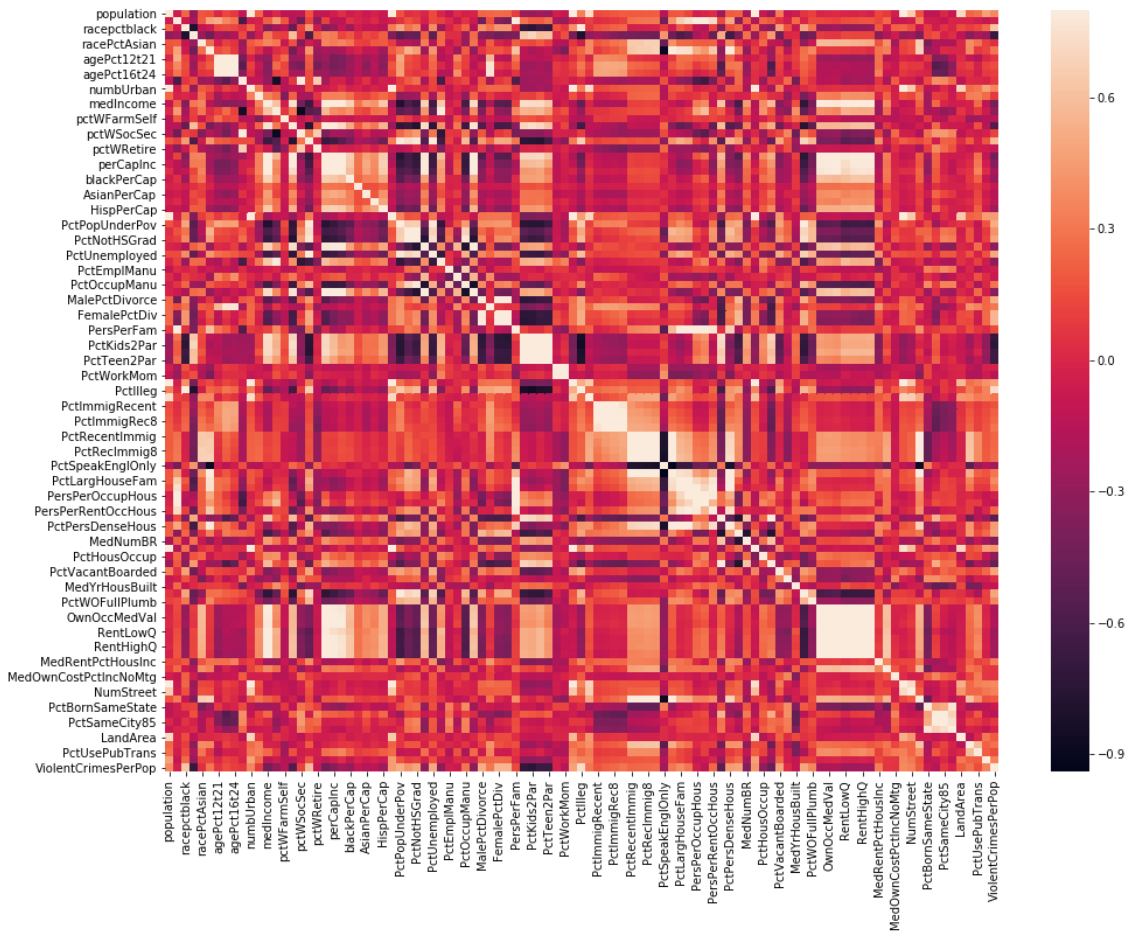


Figure 7 Correlation matrix

In this case, the output is a regression value so using accuracy is nonsense. However, mean squared error score has the capacity of evaluating the results of the model so we are going to use it to evaluate each one of the splits.

Next graphs show the validation loss function as a function of epochs, there is one plot for each k fold.

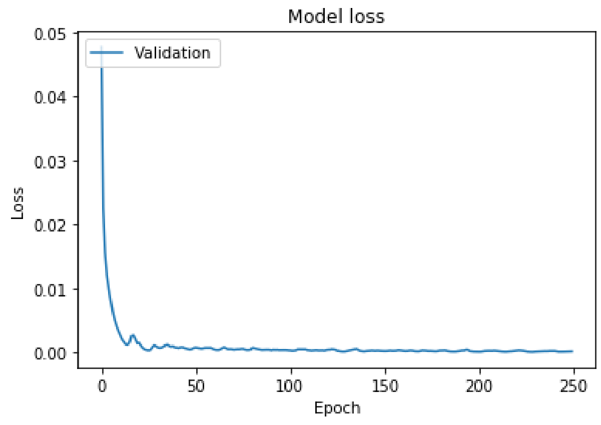


Figure 9 mean_squared_error: 1.93%

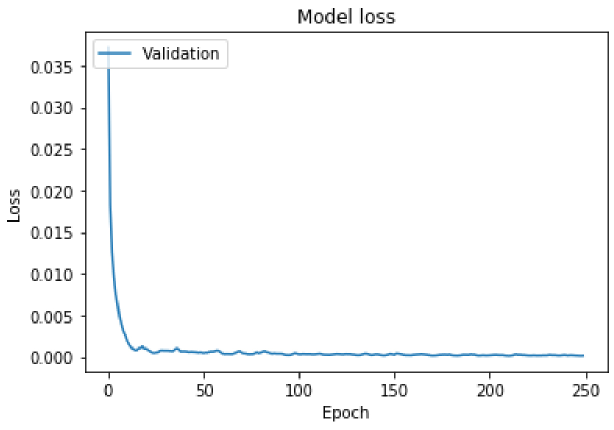


Figure 8 mean_squared_error: 1.75%

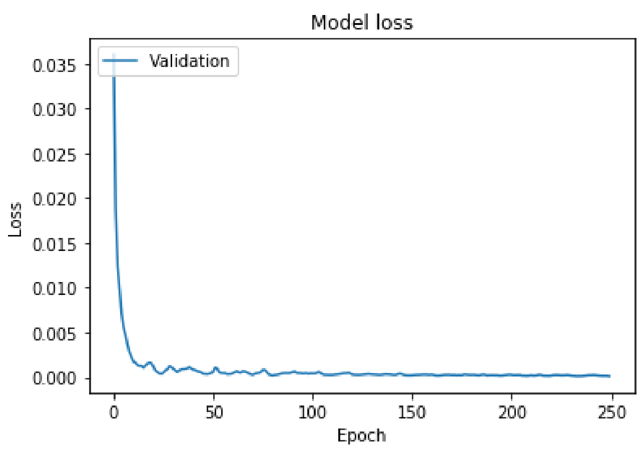


Figure 11 mean_squared_error: 2.00%

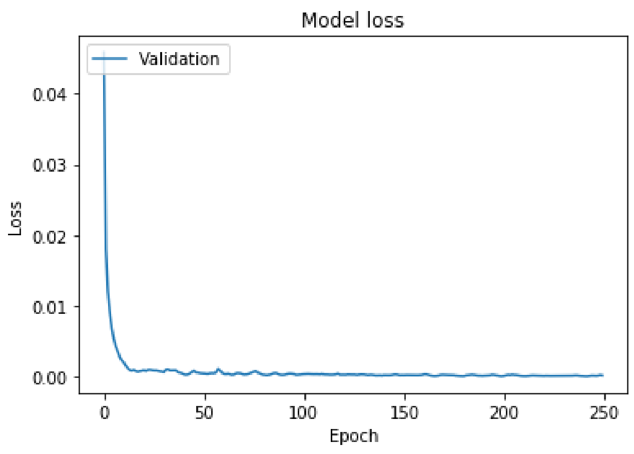


Figure 10 mean_squared_error: 2.09%

Taking the average of mean squared errors, we have 1.94% (+/- 0.13%) in total as the score of the average of the validation data. Next, we will fit the model with all the training data once and evaluate it with the testing set.

Evaluating in the test set, we get a mean squared error of 2.12% and the loss is plotted in the next graph:

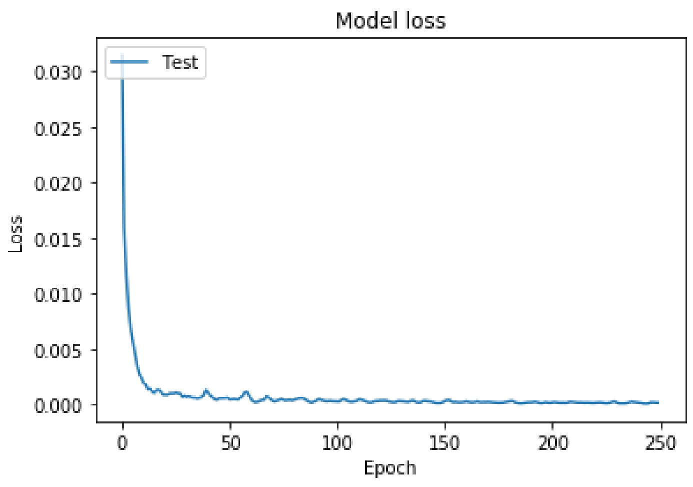


Figure 12 Test set loss

Discussion

Those three tasks let us learn several facts about dep learning, here are some:

1) In task 1, fully connected layers perform quite well, however, the computation cost of the training is very high. In the case of inputs being images, convolutional neural networks perform much better. CNN's that have less weights to train than conventional NN's (in NN all the neurons are connected, that's a lot of connections) usually gets better accuracy in this kind of problems.
2) Adding regularization to training makes the process slower but helps not to overfit the training data and the regularization parameter cannot be very high because then training would be slower.
3) In task 2, validation accuracy seems to be higher than training accuracy, this can be because the size of the validation set is very small compared to the training set and for some reason it is getting higher scores.
4) In task 3, we have a lot of features and one option for future work could be to use principal component analysis to reduce the dimensionality and that way, we would have similar information with less features.

References

1) https://keras.io/examples/cifar10_cnn/

2) https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/